AD-A073 879 ARMY WAR COLL CARLISLE BARRACKS PA F/G 9/2

A REVIEW OF ARMY HIGH LEVEL PROGRAMMING LANGUAGES AND DOCUMENTA--ETC(U)

MAY 79 J G MCKNIGHT

UNCLASSIFIED NL

1 OF 1

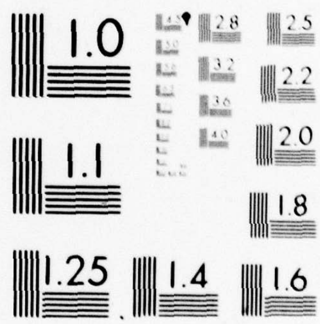
AD A073879

END

DATE FILMED

10-79

DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL

STUDY PROJECT

26 MAY 1979

A REVIEW OF ARMY HIGH LEVEL PROGRAMMING LANGUAGES AND DOCUMENTATION FOR ADP SOFTWARE

by

Lieutenant Colonel James G. McKnight
Quartermaster Corps

US ARMY WAR COLLEGE, CARLISLE BARRACKS, PA 17013

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

6

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

A REVIEW OF ARMY HIGH LEVEL PROGRAMMING LANGUAGES AND DOCUMENTATION FOR ADP SOFTWARE

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

LTC James G. McKnight

10

8. CONTRACT OR GRANT NUMBER(s)

12 49p.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

US Army War College
Carlisle Barracks, PA 17013

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Same as Item 9

12. REPORT DATE

11 26 May 1979

13. NUMBER OF PAGES

41

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

9 Study project,

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper reviews state of the art for high level programming languages and traces their development from first generation computers to the present day. A review and comparison of industry and Army standards for program documentation is made and differences discussed. Based upon field interviews, findings concerning command unique documentation are presented. Conclusions indicate that more powerful and cost effective high level languages are

403 565

available for use and that documentation for command unique software is not adequate. Recommendations are made for securing a system language, establishing a central review and approval authority for software documentation, and for various follow up studies.

USAWC MILITARY STUDIES PROGRAM PAPER

A REVIEW OF ARMY HIGH LEVEL PROGRAMMING LANGUAGES AND DOCUMENTATION FOR ADP SOFTWARE

by

Lieutenant Colonel James G. McKnight
Quartermaster Corps

US Army War College
Carlisle Barracks, Pennsylvania 17013
26 May 1979

AUTHOR(S): James G. McKnight, LTC, QMC

TITLE: A Review of Army High Level Programming Languages and Documentation for ADP Software

FORMAT: Individual Study Project

DATE: 26 May 1979 **PAGES:** **CLASSIFICATION:** Unclassified

This paper reviews state of the art for high level programming languages and traces their development from first generation computers to the present day. A review and comparison of industry and Army standards for program documentation is made and differences discussed. Based upon field interviews, findings concerning command unique documentation are presented. Conclusions indicate that more powerful and cost effective high level languages are available for use and that documentation for command unique software is not adequate. Recommendations are made for securing a system language, establishing a central review and approval authority for software documentation, and for various follow up studies.

TABLE OF CONTENTS

Chapter I

PROBLEM STATEMENT AND METHODOLOGY

The purpose of this review is to evaluate "high level" programming languages used within the Army and to determine if state of the art capability is being exploited. In addition, a review of existing software documentation standards will be conducted in order to determine adequacy.

Although numerous high level programming languages exist within the ADP community, the U. S. Army currently limits ADP languages to three: FORTRAN, COBOL, and BASIC. (Exceptions exist for limited purposes.) These languages are in use due to historic situations and may or may not present limitations to programming capability. Closely related to program languages is program documentation, with some languages being more self documenting than others. Long term benefits can be expected when documentation is properly accomplished. Improper documentation can lead to excessive costs when specific programmers change employment, when new hardware is purchased or when attempts are made to export the system.

In order to narrow the scope of the review, and to keep it within manageable bounds, only command unique software will be addressed. This limitation also recognizes the fact that Department of the Army standard systems are developed by organizations with greater resources and capabilities than local data processing institutions.

Methodology for conducting this study consisted of a review of applicable Department of Defense instructions, Department of the Army Regulations, U. S. Army Computer Systems Command Technical Manuals, academic literature and personal interviews with military and civilian programmers and managers.

Department of Defense instructions, Department of the Army Regulations, and U. S. Army Computer Systems Command Technical Manuals were used to establish familiarity with existing standards and to provide a base for comparison. Likewise, academic literature was used to gain knowledge of standards accepted by the teaching profession. The personal interviews allowed a determination of "in fact" situations and comparisons against standards to be made. In the interest of accurate information, those interviewed were given assurances of non-attribution. Locations selected for obtaining interviews were wide spread and diverse. In all, thirty programmers from two divisions, one Corps headquarters, an Army headquarters and two installations were interviewed. In addition, management personnel from Department of the Army, U. S. Army Computer Systems Command and the Army and Air Force Exchange System were contacted.

Interviews and discussions were based upon the following questions:

1. In what way did you learn to program?

2. What is your experience level?

3. In what programming languages are you qualified?

4. When programming, how do you select variable names to be used?

5. When programming, how do you determine the composition of a data element.

6. Who is responsible for software documentation?

7. Does your software documentation contain a program narrative, comments throughout the program, and a program flowchart?

8. Who approves your finished documentation?

9. In your opinion, and based upon your experience, have your supervisors been technically capable of reviewing your work?

10. When programming, what reference documents do you use to assist you should you have problems?

11. Do you program for hardware independence?

12. Has any outside agency ever checked your completed work?

13. How do you account for time spent programming?

14. Are you given a restriction as to the maximum allowable core size which you may use?

15. When programming, do you use subroutines? If so, how?

16. Do you have any comments concerning program languages or documentation which you would like to make?

Certain data processing terms, which are important for accurate understanding, are used throughout the text of this report. Definitions are therefore provided below. Unless otherwise stated, definitions are from IBM's "Data Processing Glossary".[1]

ANSI: American National Standards Institute. An organization sponsored by the Business Equipment Manufacturers Association (BEMA) for the purpose of establishing voluntary industry standards. Abbreviated ANSI.

Assembly: (assembler language) A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer.

BASIC: An algebra-like language used for problem solving by engineers, scientists and others who

may not be professional programmers.

Character: A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes.

COBOL: COmmon Business-Oriented Language. A business data processing language.

Command Unique: An ADP system supporting a functional application unique to a single Army command, installation, or activity. (Author's defination.)

Comment: (comment statement) A statement used to include information that may be helpful in running a job or reviewing an output listing.

Core: (magnetic core) A configuration of magnetic material that is, or is intended to be, placed in a spatial relationship to current-carrying conductors and whose magnetic properties are essential to its use. It may be used to concentrate an induced magnetic field as in a transformer induction coil, or armature, to retain a magnetic polarization for the purpose of storing data, or for its nonlinear properties as in a logic element. It may be made of such material as iron, iron oxide, or ferrite and in such shapes as wires, tapes, toroids, rods, or thin film.

Data Element: Grouping of information units which have a unique meaning and sub-categories (data items) of distinct units or values. Examples of data elements are military personnel grade, sex, race, geographic location, and military unit. (Army Regulation 18-1)

Flowchart: A graphical representation for the definition, analysis, or solution of a problem, in

which symbols are used to represent operations, data, flow, equipment, etc.

FORTRAN: FORmula TRANslating system. A language primarily used to express computer programs by arithmetic formulas.

Hardware: Physical equipment, as opposed to the computer program or method of use, for example, mechanical, magnetic, electrical, or electronic devices.

High Level Language: A language used to prepare computer programs. A set of representations, conventions, and rules used to convey information; more like human language than machine language. A language which must be translated into machine language prior to use by a computer.

JOVIAL: A high level language used within DOD for programming command and control applications. (Author's definition)

Listing: (program listing) A printout, usually prepared by a language translator, that lists the source language statements and contents of a program.

Overlay: (1) The technique of repeatedly using the same blocks of internal storage during different stages of a program. When one routine is no longer needed in storage, another routine can replace all or part of it. (2) A program segment or phase that is loaded into main storage. It replaces all or part of a previously loaded segment.

PL/1: A high-level programming language, designed for use in a wide range of commercial and scientific computer applications.

Program Narrative: The details and characteristics of a program or subroutine which would be of value to a maintenance programmer in understanding the program.

RPG: Report program generator.

Software: A set of programs, procedures, and possibly associated documentation concerned with the operation of a data processing system. For example, compilers, library routines, manuals, circuit diagrams.

Subroutine: A segment of coded instructions which can be stored at one place and can be linked to one or more other calling routines.

Variable Name: An alphameric term that identifies a data set, a control statement, a program, or a procedure.

Word: A sequence of bits or characters treated as a unit and capable of being stored in one computer location. Synonymous with machine word.

Chapter II

PROGRAMMING LANGUAGES

The history of programming languages parallels the development phases as the computer itself. Original first generation computers were slow, had limited capacity and very little capability for control or assistance to programs being used. The programmer had to specify in exact and painstaking detail each function to be performed. In order to do so, the programmer used either machine or assembly language. A thorough knowledge of hardware was required and programs were complex. As an example, consider the steps necessary to keep track of an exponent without the aid of floating point arithmetic now so commonly used.

As computer capability passed into the second and third generation, more machine assistance became available. The programmer no longer had to program in an abstract form which might read M333272P (Move contents of location 333 to location 272 and punch a card.). Rather, high level languages were available. High level languages are more English-like or are closely related to mathmatical formulas. The programmer does not have to keep track of finite details and can program in broader terms. In turn, the computer uses a software routine called a compiler to transform high level languages into machine languages. The power of this capabilty can be seen in the fact that one high level instruction, after being compiled, may result in an entire subroutine of machine language code. First generation languages (assembly) resulted in a one for one translation. High level languages also have the major advantage of allowing the programmer to think in a language closely related to normal problem solving. Well known examples are COBOL

and FORTRAN.

A new capability is being seen in the market place today. This capability is high level language which "designs" a system and is naturally referred to as "system language". Just as a COBOL statement generates multiple machine language instructions, system language instructions generate multiple COBOL statements which then generate multiple machine instructions. As one example, a system language program consisting of 162 instructions generated a COBOL program of 570 statements! Not only were the statements generated, but they were error free. Estimates also indicate that 75% of direct programming time can be saved and of even greater importance is the fact that a system language generates the total documentation necessary for supporting the software.

To further illustrate the capability of this type of high level language, a detailed description of one such language will be give. Through the courtesy of Thorne Data Processing, Springfield, Virginia, SL/1 is described.

"SL/1 is a very high-level language. As its title implies, it is a systems language. It is not a programming language, but rather a systems language used to specify the data elements for each input and output of a system. The language is very powerful and permits the user to completely define the total requirements of a system in terms of elements associated with inputs and outputs. These elements may be input elements or generated elements for which the user must provide the element formulas. Full arithmetic and conditional logic may be used in specifing the element

formulas. The system elements provide the basic ingredient for the building block approach that is used; the subordinate elements in the element formulas are used to build the basic elements, the basic elements are used to build the inputs and outputs, and the inputs and outputs are used to build the system. Since SL/1 is a data and element-formula specification language, it does not contain the verbs and operations associated with conventional programming languages, i.e., READ TO GO, etc. The SL/1 user is never concerned with such programming details.

During the Requirements Study the analyst used the SL/1 language to record data related to each data input (initial inputs) and each data output (final product) of the proposed system. For inputs, the analyst records the elements (by name) appearing on the inputs along with element characteristics (size, class, number of decimals, and whether it has a sign). Also, information related to retention (whether or not the element should be retained in a master file) of the elements is recorded. For outputs, the element names desired on each output, their characteristics, and where they print (both detail print information and accumulated totals) is recorded. Report title/header line constants and total line identification constants are also recorded. The formulas of elements to be generated by the system in association with either inputs or outputs are given. Only the simple element name relationships using conditional and/or arithmetic logic are given.

Programs are not defined. Master and work files are not defined. Only the system inputs and outputs

are defined. While SL/1 takes care of the design and programming details automatically, through training and application users learn how SL/1 implicitly reacts to logical system specifications and are able to predict or force resultant physical system configurations. The analyst describes a data flow pattern using the SL/1 language. The analyst essentially states at a very high level WHAT is to be accomplished and SL/1 does the burdensome detail work. The SL/1 language statements are free form. Therefore, any 80 character coding sheet, such as a COBOL coding form, may be used to record the statements."

Documentation provided by SL/1 includes:

Input/Output system specifications
Report facsimilies
Documentation cover page
Table of contents
System description
List of system inputs
Expanded list of system inputs with descriptions
List of system outputs
Expanded list of system outputs with descriptions
Master file(s) explanation
Master file(s) general information
General narrative of programs
Detail narrative of each program
Operation of system narrative
Production control sheet
Tape librarian guide
Key punch instructions
System flowchart
Run Book (operating documentation)
JCL cards
Generation data set control JCL
Program input/output record layouts

Element to record layout cross-reference
Element dictionary
Element edit dictionary
Element edit diagnostic dictionary
Master file record(s) COBOL specifications
Code Book (for elements that are codes)
Appendices
Source program listings
Master file punch outs

Chapter III

SOFTWARE DOCUMENTATION

In discussing the requirements and accepted principles of documentation, a clear distinction must be made between system documentation and software documentation. The first, and most obvious, distinction being that software documentation is a sub-element of system documentation. Documentation for an ADP system includes the information and instructions necessary for overall system usage and understanding. As software is a narrow sub-set of an overall system, it follows that software (also referred to as a program) documentation addresses a much narrower area. The Department of Defense describes system documentation as being necessary in order to:

"a. Provide managers with documents to review at significant developmental milestones to determine that requirements have been met and that resources should continue to be expanded.

b. Record technical information to allow coordination of later development and use/modification of the ADS.

c. Ensure that authors of documents and managers of project development have a guide to follow in preparing and checking documentation.

d. Provide uniformity of format and content of ADS documentation throughout DOD components."[2]

Industry tends to narrow it's definition of system documentation somewhat, since it does not have to worry about various components such as those found within the Department of Defense. One definition of system documentation applicable to industry (and other users as well) reads:

> "System documentation encompasses all information needed to define the proposed data processing system to a level that it can be programmed, tested, and implemented. The major document is a system specification which acts as the permanent record of the structure, functions, flow, and control of the system. It is the basic medium of communication of information about the proposed system between the systems design, programming, and user functions. System documentation thus specifies how a system will operate."[3]

Contrasting the above system documentation reasoning to the specifics of software documentation, a summary of software documentation can be expressed as "documentation comprises the records of the detailed logic and coding of the constituent programs of a system."[4]

With the distinction between system and software documentation in mind, a closer look at reasons for software documentation should be provided. Several reasons exist, but most can be classed either directly or indirectly into the broad area of maintenance. Maintenance, in this case, being simply those actions, changes, modifications etc. necessary to keep an operational software program current with user needs or operating in a changing hardware environment. The exception to this classification is documentation necessary to aid in program development and acceptance. The consideration in software maintenance therefore is, or should be, dollar cost. The direct relationship being a high cost when documentation is poor. Whatever the reason, time wasted by a programmer or analyst

translates into dollars.

Perhaps the most basic element of program maintenance is trouble shooting. This can occur when a system is placed prematurely into an operational mode, when data with unplanned paramenters becomes the norm, or when a system is being field tested. Another maintenance aspect occurs when hardware is changed or converted, as different manufacturers have different software capabilities built into their computers. This is of great importance in the military community as competitive procurement is mandated. Mobility of the military member also directly affects software maintenance. Seldom, if ever, is the programmer's availability the same as the life cycle of the software. To the extent that documentation is accurate, another programmer will be able to answer questions and perform maintenance upon the program. Again, poor documentation results in later unnecessary costs. Unfortunately these costs are normally tied to "overhead" or "operating costs" and are not attributed to the true cause.

Although different elements of industry have different software documentation standards, most generally follow the following outline:

1. System Flow Chart: shows relation of specific program to entire system.
2. System Narrative: similar to system flow chart, but narrative rather than visual.
3. Program Flowchart: visual representation of program logic and structure.
4. Program Narrative: same as system narrative, but pertaining only to the software.
5. Commented Listing: a listing of computer

program instructions, produced by the computer, with explanations of all variables and methods. It is often linked to specific areas of a flow chart by numbers.

6. File and Record Description: information describing input and output configuration.

7. Set-up and Run Procedures: information describing how to load and operate the software on a computer. Also includes instructions for error conditions.

8. Discussion of Valid Data Ranges: an explanation of any limits existing within the program. An example might consist of a routine which would check for maximum dollar value for an order.

9. Discussion of Algorithms: used when complex sorting, formulas, or procedures exist.

The composition of documentation for Army software as expressed in Army Regulation 18-1,[5] is: Narrative system description, system description, system flow chart, equipment and software requirements, list of programs, program description, program run diagram, input device document, output form layout and samples, file and record layouts, detained program narrative, processing macro-logic chart, decision tables, timing criteria, operating instructions, memory layout, detail program flowchart, program logic details, miscellaneous information, program compilation output, test data and criteria test output results, and test timing results.

Although the Army's criteria for software documentation appears to be more detailed, a closer examination reveals that several components

of the Army's standards are in fact expressed as one industry standard. For example, the Army's requirement for: equipment and software requirements, program run diagram, input device document, output form layout, timing criteria, operating instructions and miscellaneous information might well be components of industry's Set-up and Run Procedures. In both cases, however, the intent is to insure that sufficient information is provided.

Two differences exist and should be recognized. Industry's requirement for comments within the program listing and for a discussion of Algorithms are not expressed in Army requirements. Without such a stated requirement, complex routines may have no recorded definition of logic used and program listings can become a maze of unknown variable names (also called data items). The Army's principle producer of large software systems, Computer Systems Command, recognizes the importance of this area and offers the following as a guide but not as a directive. "Self documenting programs aid in developing, maintaining and reusing programs. The most important and, unfortunately, the most neglected program documentation is the description of the meaning and usage of data items. How often have we questioned why a programmer redefines an alphanumeric data item as numeric? We wonder why the programmer did not choose one class or the other; so we go tripping through the program looking for the answer. The easy solution to this problem is to provide a brief comment by the data item description."[6]

To the extent that standards are not clearly stated and, most importantly, understood, the

unfortunate programmer and supervisor are placed into a position of asking: What do I record?, When do I record it?, How much detail?, and How will it be used? For the Army, one other important question must be answered. This question, "Who approves the program documentation and determines that it is sufficient?", has no apparent answer.

Chapter IV

FINDINGS

A detailed examination and discussion of data collected during interviews will be presented on this and following pages. Conclusions and recommendations are stated in Chapter V. Discussions are related to each question of the interview.

Question 1. "In what way did you learn to program?"

The purpose of this question was to provide a basis for evaluating the mix of instructional bias existing among those being interviewed. For example, if a high percentage of those interviewed learned in a manufacturer sponsered environment, then bias toward specific techniques and standards would exist. Responses indicated that a bias should not be expected, as 43% had learned in various colleges and civilian institutions, 40% from formal Army training, and 17% from on job training within the Army or from self taught methods. It therefore appears that a valid random selection of persons to be interviewed was made.

Question 2. "What is your experience level?"

This question was used to gain further insight into the information gathered in later questions. A total of 145 years experience, averaging 4.8 years per programmer was found. Of equal importance, however, is the fact that these programmers have served in 29 locations world-wide. It can therefore be reasonably assumed that their experiences are representative of overall conditions throughout the Army.

Question 3. "In what programming languages are you qualified?"

This question provides a basis for evaluating

comments and experiences about documentation, as some languages are more self-documenting than others. Also, knowledge of languages provides a basis for discussing existing Army standards. Responses indicated that all were familiar with COBOL, 60% knew FORTRAN, 46% could program in ASSEMBLY, 26% in BASIC and RPG, 20% were capable in PLI and 10% in other languages such as PLC and JOVIAL. Only 26% of the programmers were knowledgable in only one language, while 20% could use two languages, 40% were capable of programming in more than three and 13% had the capability to program in more than five languages.

Question 4. "When programming, how do you select variable names?"

Responses to this question indicate the degree to which standard conventions are used. Responses indicate that selection was entirely up to the programmer. All programmers, with one exception, indicated that the same basic process was used--it being the selection of "meaningful" or "logical" names. The obvious problem being that a name which is meaningful or logical to one person may or may not be to another. The exception mentioned above indicated that he used "the Computer Systems Command Naming Convention as outlined in their manuals".

Question 5. "When programming, how do you determine the composition of a data element?"

Command unique software often becomes the basis for Department of the Army Standard Systems and is also frequently exported to other locations. To the extent that standard data elements are used, as shown in the 18-12 series of Army Regulations (Catalog of Standard Data Elements and Codes),

interface conditions are quaranteed. As a simplistic example, consider a program which has as required output military grades. Should "Captain" be expressed as "CPT", "Capt", "03" or "Captain". To the extent that 18-12 is followed, the potential for reprogramming is avoided. Obviously, the longer the program, the more reprogramming/cost involved. Responses to this question indicated that programmers are not aware of the 18-12 series of Army Regulations and that designers of command unique systems often leave details of data elements to the programmers decision. Not one programmer indicated that reference was made to formal publications when the need to determine a data element arose. Rather, a "meaningful term" was used or a "whatever is logical" type decision was made. These types of decisions appear to be frequently made by programmers, as 67% indicated that specifics were not provided by designers. Expressed in terms of the designer, only 33% provided details. In relation to the "Captain" example given earlier, the programmer might have detailed instructions on data manipulation, but only broad guidance as to input/output data element construction.

Question 6. "Who is responsible for documentation?"

This question relates to the specific area of program documentation as opposed to the broader aspect of system documentation. The question was asked in order to determine the degree of programmer independence. Responses indicated that the programmer is 100% responsible.

Question 7. "Does your software documentation contain a program narrative, comments throughout the program, and a flowchart?"

This is perhaps the most complex question asked during the interviews and provoked the most discussion of all questions. Accordingly, detailed discussions of principles involved will be provided. To the portion of the question concerning a program narrative, 87% responded that a narrative was used. This figure can be further divided and indicates that 46% use a narrative within the program listing and 41% restrict narratives to the overall documentation package. Comments were used by 46% of the programmers, while 33% indicated that flowcharts were used. Although somewhat lengthy, consider the following example of a small subroutine coded in FORTRAN. Without comments or narrative, the code looks like:

```
      SUBROUTINE ABLE(IARRAY,IROW,ICOL,ICSSEQ,ISORT)
      DIMENSION IARRAY(25,25),ICSSEQ(25),IOUT(20)
      NTHRU=IROW-1
      DO 20 ITIMES=1,NTHRU
      IPOINT=ITIMES
      ICARRY=ITIMES
      DO 30 IR=ICARRY,IROW
      DO 40 KOLUMN=1,ISORT
      ICOLSB=IABS(ICSSEQ(KOLUMN))
      IF(ICSSEQ(KOLUMN).GT.0) GO TO 35
      IF (IARRAY(IPOINT,ICOLSB).LT.IARRAY(IR,ICOLSB))
      GO TO 45
      IF(IARRAY(IPOINT,ICOLSB).EQ.IARRAY(IR,ICOLSB))
      GO TO 40
      GO TO 30
   35 IF(IARRAY(IPOINT,ICOLSB).GT.IARRAY(IR,ICOLSB))
      GO TO 45
      IF(IARRAY(IPOINT,ICOLSB).LT.IARRAY(IR,ICOLSB))
      GO TO 30
   40 CONTINUE
   45 IPOINT=IR
   30 CONTINUE
      DO 50 ISWITCH=1,ICOL
      ISTORE=IARRAY(IPOINT,ISWICH))
      IARRAY(IPOINT,ISWICH)=IARRAY(ITIMES,ISWICH)
      IARRAY(ITIMES,ISWICH)=ISTORE
   50 CONTINUE
   20 CONTINUE
      RETURN
      END
```

Although the routine has but 26 instructions, it is extremely difficult to decipher without additional information. One can easily see the difficulty facing a programmer when tasked to modify such a program. The complex situation existing in a 2000 instruction code sequence would require tremendous expenditures of time before modification could begin. Now, consider the same sequence of code when comments are added:

```
      SUBROUTINE ABLE(IARRAY,IROW,ICOL,ICSSEQ,ISORT)
C
C     THIS SUBROUTINE PERFORMS THE ACTUAL SORT
C     ROUTINE.  THE COLUMN SORT SEQUENCE DETERMINES
C     THE COLUMNS TO BE SORTED AS WELL AS THE SEQUENCE.
C     A + INDICATES ASCENDING AND A - DECENDING
C     VARIABLE NAMES
C     NTHRU-NUMBER OF TIMES THROUGH THE ARRAY
C     ITIMES-CONTROLS MOVEMENT THROUGH DATA TO
C     BE SWITCHED
C     IPOINT-BASE COMPARE LOCATION
C     ICARRY-'SEARCH' COMPARE LOCATION
C     ICOLSB-USED FOR COLUMN IDENTIFICATION
C
C     DO LOOP 20*****DETERMINES HOW MANY TIMES
C                    THE SWITCHING WILL OCCUR.
C                    (NUMBER OF ROWS TO BE SWITCHED
C                    -1)
C     DO LOOP 30*****CONTROLS THE ROW SELECTION
C                    FOR COMPARISON WITH POINTER.
C     DO LOOP 40*****CONTROLS THE COLUMN BEING
C                    USED WITH THE COMPARE SEQUENCE.
C                    BY MOVING THROUGH THIS LOOP,
C                    WITHOUT ENTERING FROM LOOP 30,
C                    A FURTHER ATTEMPT TO RESOLVE
C                    EQUAL SITUATIONS CAN BE MADE.
C     DO LOOP 50*****CONTROLS THE ACTUAL SWITCHING
C                    OF ARRAY ELEMENTS, AND DUE
C                    TO ITS LOOP CAPABILITY, THE
C                    ULTIMATE SWITCHING OF ENTIRE
C                    ROWS WITHIN THE ARRAY.
      DIMENSION IARRAY(25,25),ICSSEQ(25),IOUT(20)
      NTHRU=IROW-1
      DO 20 ITIMES=1,NTHRU
      IPOINT-ITIMES
      ICARRY=ITIMES
      DO 30 IR=ICARRY,IROW
      DO 40 KOLUMN=1,ISORT
```

```
C
C   INSURE POSITIVE NUMBER
C
      ICOLSB=IABS(ICSSEQ(KOLUMN))
C
C   CHECK FOR ASCENDING ORDER-BRANCH TO 35 IF
C   POSITIVE
C
      IF(ICSSEQ(KOLUMN).GT.0) GO TO 35
C
C   CHECK VALUES-BRANCH AS APPROPRIATE
C
      IF (IARRAY(IPOINT,ICOLSB).LT.IARRAY(IR,
      ICOLSB)) GO TO 45
      IF(IARRAY(IPOINT,ICOLSB).EQ.IARRAY(IR,
      ICOLSB)) GO TO 40
      GO TO 30
C
C   CHECK VALUES-BRANCH AS APPROPRIATE
C
   35 IF(IARRAY(IPOINT,ICOLSB).GT.IARRAY(IR,
      ICOLSB))GO TO 45
      IF(IARRAY(IPOINT,ICOLSB).LT.IARRAY(IR,ICOLSB))
      GO TO 30
   40 CONTINUE
C
C   RESET POINTER
C
   45 IPOINT=IR
   30 CONTINUE
C
C   SWITCH ELEMENTS
C
      DO 50 ISWICH=1,ICOL
      ISTORE=IARRAY(IPOINT,ISWICH)
      IARRAY(IPOINT,ISWICH)=IARRAY(ITIMES,ISWICH)
      IARRAY(ITIMES,ISWICH)=ISTORE
   50 CONTINUE
   20 CONTINUE
      RETURN
      END
```

An observer need not be knowledgable of FORTRAN to see that the addition of comments has greatly added to the understanding of the subroutine. By looking at comments pertaining to the overall program, further understanding can be achieved. In this illustration, such comments might read: (Comments would appear in the program listing.)

```
C
C   THIS IS A FORTRAN PROGRAM WHICH CONTAINS
C   A SUBPROGRAM.  THE PURPOSE OF THE PROGRAM,
C   THROUGH USE OF THE SUBROUTINE, IS TO PERFORM
C   A TWO DIMENSIONAL,MULTI-COLUMN,VARIABLE
C   SEQUENCE SORT.  IT ALSO UTILIZES VARIABLE
C   INPUT AND OUTPUT FORMATS.(THE PROGRAM HAS
C   NOT BEEN MODIFIED SINCE IT WAS WRITTEN)
C   PROGRAM VARIABLES
C       ALPHA-ALPHANUMERIC INFORMATION
C       ICSSEQ-COLUMN SORT SEQUENCE
C       IN-INPUT FORMAT
C       IOUT-OUTPUT FORMAT
C       IARRAY-ARRAY FOR HOLDING DATA
C       IROW-NUMBER OF ROWS OF DATA
C       ICOL-NUMBER OF COLUMNS OF DATA
C       ISORT-NUMBER OF COLUMNS TO BE SORTED
C
```

An overall relation has now been provided. Again, consider a complex 2000 instruction sequence of code and consider the addition of a program narrative:

PROGRAM NARRATIVE DESCRIPTION

This is a Fortran program, which contains a main program with a subroutine call. It performs a two-dimensional, multi-column, variable sequence sort, and utilizes variable input and output formats. The maximum number of rows and columns to be sorted must be equal to or less than 25. In order for the program to function properly, the following data must be supplied:

Number of rows of data
Number of columns of data
Number of columns to be sorted
Problem description
Column sort sequence with indication of
ascending or descending
Input format for data
Output format for data

Note: This program is an independent program, and does not interface with any system.

The operations are as follow:

1. The number of rows and the number of columns in the data set is read, along with the number of columns to be sorted. Alphanumeric information describing the problem is also read.

2. Formats to be used for input and output are read.

3. All of the information listed above is printed, with appropriate labelling information.

4. Data is read and printed in its original sequence.

5. A subroutine (Able) is used to perform a sort of the data, using the "Modified Cascade method.

6. Data is printed in its sorted sequence.

7. If additional data is present, steps 1-6 are re-executed

8. The program terminates on an error condition when additional data is not present.

A programmer or analyst with basic FORTRAN knowledge would now have a much better concept of program logic and intent. To give a visual assist, a flowchart (Figure 1) can be provided. From the above example, it can be seen that the absence of either narrative, flowchart, or comments degrades the capability for program maintenance. The absence of all three can create an impossible situation.

Question 8. "Who approves your finished documentation?"

Answers to this question indicate the degree to which documentation is reviewed for adequacy. Opinions as to capabilities of those reviewing are addressed in Question Nine. Seventy-three percent responded that their supervisor/section chief

SUBROUTINE ABLE

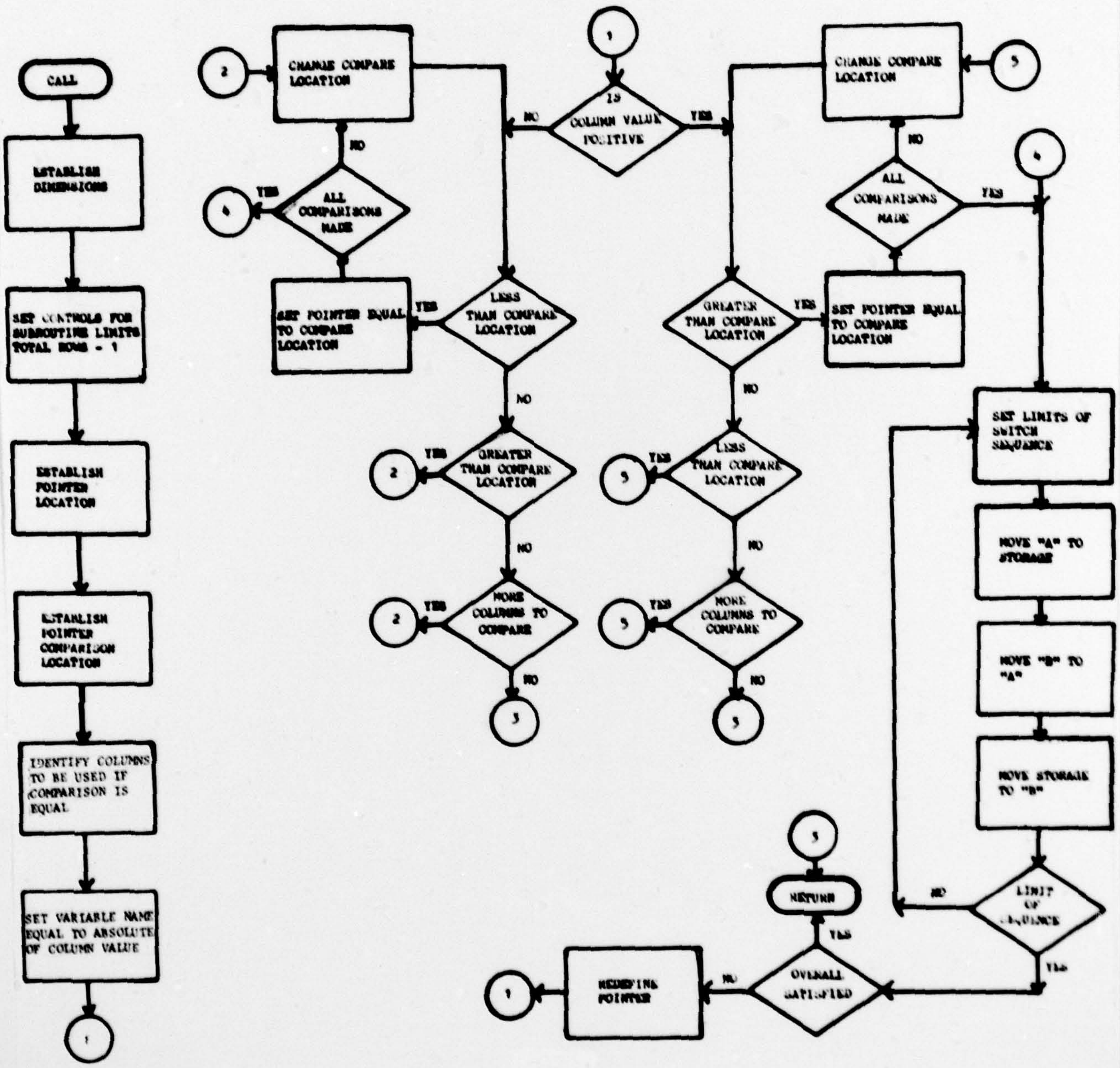


FIGURE 1

approved the documentation. The disturbing fact remaining is that 27% indicated that no approval was required.

Question 9. "In your opinion, and based upon your experience, have your supervisors been technically capable of reviewing your work?"

This question is significant in relation to question eight and also provides insight into the independence of programmers working in a command unique environment. Fully 80% expressed the opinion that their supervisors were not technically capable of providing assistance or evaluating work in the area of programming. This perception related almost exclusively to senior supervisors in the grade of E-7, E-8, E-9, and to all officers. The inability of senior enlisted members to deal with details of programming seems to stem from previous promotion patterns. Although not a specific interview question, it seems that many enlisted supervisors reached their present positions through the route of machine operator. It follows then, that their knowledge of programming would be limited. Lack of officer capability seems linked to the wide background of those possessing specialty code 53 (ADP). This specialty code is used to identify officers possessing everything from on job experience to doctorial degrees. Some may well be qualified to deal with programming, but the perception is that most are not.

Question 10. "When programming, what reference documents do you use to assist you should you have problems?"

Specific guidelines are available to programmers in technical manuals published by the U. S. Army Computer Systems Command. Use of these manuals

would insure maximum adherence to Army and ANSI standards and would impact specifically upon the use of manufacturers unique capabilities. The important factor being that standards tend to insure software compatibility among various hardware configurations. Some examples of manufacturer's unique features include Burrough's capability to index more than two fields and the use of "vs" for designating literals. Control Data Corporation offers a six character rather than a four character word configuration, and IBM's "Transform" verb is unique. The use of these types of unique capability obviously tie a program to specified hardware and require expenditures of time and money when hardware is changed. It was found that manufacturer's literature was used as the sole reference by 73% of the programmers, 20% used Army publications, and 7% used a mix of both Armyand manufacturer supplied documents.

Question 11. "Do you program for hardware independence?"

This question is related directly to question ten and directly impacts upon the area of software conversion cost. The fact that 87% of those interviewed responded "no" indicates that programmers either are not aware of, or are not concerned with, possible conversion to different hardware at a later date.

Question 12. "Has any outside agency ever checked your completed work."

In 93% of the responses, the answer was no. Considering that work may have been reviewed after a programmer's departure, it could be argued that this figure is high. However, only 13% indicated

that an outside agency (Inspector General or Command Management Information System Office) had ever been observed to review documentation. Indications are that external review agencies simply do not have the capability to review documentation for adherence to requirements and for adequacy.

Question 13. "How do you account for time spent programming?"

This question was designed to give an insight into how developmental costs are accumulated on command unique software. Without an accurate accounting of time, a large portion of cost will not be reflected. In turn, it becomes impossible to know when Department of the Army standards for command unique software/systems have been exceeded. Resposes indicated that 87% had no requirement or no effective way to keep track of programming effort devoted to a particular program or system.

Question 14. "Are you given a restriction as to the maximum allowable core size which you may use?"

About half, 46%, stated that a restriction of some sort was given. Amount of core to be used varied and was dependent upon total core available. Those interviewed, who reflected no restriction, indicated that core was not a problem but that they programmed to use "as little core as possible". This question is somewhat related to question eleven and provides another indication as to how standards are being observed.

Question 15. "When programming do you use subroutines? If so, how?"

The use of subroutines, which break logical sequences of code into smaller stand alone modules, has several advantages. Of prime importance is the use of this technique to control the size of core to be used. By using subroutines of a specified

size, large programs can be produced which will run on both large and small computers. (By the use of "overlaying", subroutines are called into the core of small machines only when needed.) Answers to this question revealed that 80% of programmers use subroutines. Of those using subroutines, 96% used them for the reason stated above. The remaining 4% used subroutines only to avoid writing repetitious sections of code.

Question 16. "Do you have any comments concerning program languages or documentation which you would like to make?"

As could be expected, many comments were made. I will, however, reflect only those which have significance to this paper.

> An E-5 Programmer--"Internal documentation is critical, if it isn't in the program listing it won't be used. External documentation can never be trusted because no one is sure it is current."
>
> An E-6 COBOL programmer--"Army standards are too strict. None of the supervisors know enough to determine when they are met."
>
> An E-5 COBOL programmer--"We have a dollar cost which is hidden. It's tied directly to poor documentation."
>
> An E-6 supervisor--"Quick fixes on software always occur under short fuses and it seems to be always late at night. If documentation was in the program listing, the job would be easier."
>
> An E-5 programmer--"Once you've tried to modify someone else's program, you understand about documentation."

A W-3 supervisor: "**Most** bosses just want the program to be placed in an operational mode. They have no concept of the long range importance of documentation. Time is not allocated for documentation. As soon as a program is running, it's on to the next job!"

Chapter V

CONCLUSIONS AND RECOMMENDATIONS

From the evidence presented in this paper, several conclusions will be made and recommendations offered. In addition, the author recognizes that this study is not conclusive and that several areas deserve further study. These areas will also be shown. It should be remembered that only the command unique environment has been examined.

Conclusion 1

High level languages are available which offer significant advantages to the Army. To adapt such a language would save programming time, provide more thorough documentation, and result in long term dollar savings.

Recommendation 1

A more detailed analysis of savings to be derived from the use of a system language should be made. Supportive evidence should then be presented to the Department of Defense and approval authority for use aggressively pursued.

Conclusion 2

Software documentation for command unique systems is being accomplished in a very haphazard manner. Standards are difficult to understand, incomplete, and without effective review or approval authority.

Recommendation 2

A more simplistic set of standards for command unique software should be developed. These standards should be published as a separate reference document and should recognize the limited resources generally available where command unique systems exist. Specific requirements should include mandatory use of comments throughout the program listing and a goal of having maximum documentation resident within the program listing.

Recommendation 3

The Army should, in recognition of inadequate technical skills existing in the command unique environment, establish a central review and approval authority for software documentation. This review of documentation (revised per recommendation 2) would achieve several benefits:

A. Significant but unknown dollar savings would accrue through reduced system maintenance cost.

B. An effective and practical method of enforcing standards would exist. Inspection agencies would have only to ask for certification of documentation when inspecting. Software not having a "Certificate of Approval" would be easily identified. In turn, software reviewed by a central agency would presumably meet all standards.

C. As a by product of the review, information concerning identification and capability of command unique systems could be extracted. A truly accurate central library could then be established.

D. By including cost as a criteria, an incentive for accurate systems cost recording would be generated.

E. The empahsis on documentation, generated by a central review, would create an awareness not now present in the command unique environment.

Recommendation 4

A review of Officers holding career field 53 specialties should be made. The object of such a

review would be to determine the validity of assignment to this field. The wide variance existing between high and low degrees of technical knowledge is unfair both to the Officer and to the Army.

Recommendation 5

A review of software documentation and its importance, as taught by the Army School System, should be made.

Recommendation 6

An effective way of reflecting the true cost of system maintenance should be developed. This cost is suspected to be high, but data is not recorded.

FOOTNOTES

1. International Business Machines Corporation, Data Processing Glossary, 1972.

2 Department of Defense, Standard 7935.1-S, Automated Data Systems Documentation Standards, p. 2-1.

3. Keith R. London, Documentation Standards, p. 7.

4. Ibid.

5. Department of the Army, Regulation 18-1, Management Information Systems Policies, Objectives, Procedures and Responsibilities, pp. H-1,2.

6. U. S. Army Computer Systems Command, Automatic Data Processing Systems Development, Maintenance and Documentation Standards and Procedures Manual 18-1, Volume I General, p. 4005.

BIBLIOGRAPHY

1. Awad, Elias M. Business Data Processing, Prentice-Hall Inc., Englewood Cliffs, N. J.: 1968

2. Department of the Army. Regulation 18-1, Management Information Systems Policies, Objectives, Procedures and Responsiblities: Washington, D. C.: 22 March 1976.

3. Department of the Army. Technical Bulletin 18-111, Technical Documentation: Washington, D. C.: November 1978.

4. Department of Defense. Instruction 5000.31, Interim List of DOD Approved High Order Programming Languages: Washington, D. C. : 24 November 1976.

5. Department of Defense. Standard 7935.1-5, Automated Data Systems Documentation Standards: Washington, D. C.: 13 September 1977.

6. Fry J. and Gosden J. Survey of Management Information Systems and Their Languages. The Mitre Corporation, Washington, D. C. : May 1968.

7. Goos, G. and Hartmanis, J. Lecture Notes in Computer Science, Design and Implementation of Programming Languages. Springer-Verlag Berlin and Heidelberg, Germany: 1977.

8. Gosden, J., Bramson, S., Fry, J., Mahle, and Sternick, H. Achieving Inter-ADP Center Compatebility. The Mitre Corporation, Washington, D. C.: May 1968.

9. International Business Machines Corporation. Candidate Language Evaluation and Recommendation Report: Gaithersburg, Maryland: Undated.

10. International Business Machines Corporation. Data Processing Glossary: White Plains, N. Y.: 1972.

11. Kopstein, Felix F. Rational vs Empirical Approaches to Job/Task Descriptions for COBOL Programmers. Human Resources Research Organization, Alexandria, Virginia: June, 1970.

12. London, Keith R. Documentation Standards. Petrocelli Books, New York: 1974.

13. U. S. Army Computer Systems Command. Automatic Data Processing Systems Development, Maintenance and Documentation Standards: Volume 1, General. Ft. Belvoir, Virginia: 15 May 1975.

14. U. S. Army Computer Systems Command. Automatic Data Processing Systems Development, Maintenance and Documentation Standards: Volume II, B3500. Ft. Belvoir, Virginia: 15 March 1972.

15. U. S. Army Computer Systems Command. Automatic Data Processing Systems Development, Maintenance and Documentation Standards: Volume III, IBM System/360 DOS. Ft. Belvoir, Virginia: 15 January 1977.

16. U. S. Army Computer Systems Command. Automatic Data Processing Systems Development, Maintenance and Documentation Standards: Volume IV, IBM System/360 OS. Ft. Belvoir, Virginia: 15 July 1976.

17. U. S. Army Computer Systems Command. Programming Procedures Manual: 18-1-1, Ft. Belvoir, Virginia: July 1977.

18. U. S. Army Computer Systems Command. Structured Design Techniques and Standards Manual: 18-1-2, Test, Ft. Belvoir, Virginia: July 1977.

19. U. S. Army, U. S. Navy, U. S. Air Force and the Irvine Computer Science Department. Proceedings of the Irvine Workshop on Alternatives for the Environment, Certification, and Control of the DOD Common High Order Language: Thomas A. Standish: 1978.